PS4

Luchen Gao

2026-02-05

Due 02/07 at 5:00PM Central.

"This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: LG

Github Classroom Assignment Setup and Submission Instructions

1. **Accepting and Setting up the PS4 Assignment Repository**
    - Each student must individually accept the repository for the problem set from Github Classroom ("ps4") – https://classroom.github.com/a/hWhtcHqH
        - You will be prompted to select your cnetid from the list in order to link your Github account to your cnetid.
        - If you can't find your cnetid in the link above, click "continue to next step" and accept the assignment, then add your name, cnetid, and Github account to this Google Sheet and we will manually link it: https://rb.gy/9u7fb6
    - If you authenticated and linked your Github account to your device, you should be able to clone your PS4 assignment repository locally.
    - Contents of PS4 assignment repository:
        - `ps4_template.qmd`: this is the Quarto file with the template for the problem set. You will write your answers to the problem set here.
2. **Submission Process**:
    - Knit your completed solution `ps4.qmd` as a pdf `ps4.pdf`.
        - Your submission does not need runnable code. Instead, you will tell us either what code you ran or what output you got.
    - To submit, push `ps4.qmd` and `ps4.pdf` to your PS4 assignment repository. Confirm on Github.com that your work was successfully pushed.

Grading

- You will be graded on what was last pushed to your PS4 assignment repository before the assignment deadline
- Problem sets will be graded for completion as: {missing (0%); - (incomplete, 50%); + (excellent, 100%)}
  - The percent values assigned to each problem denote how long we estimate the problem will take as a share of total time spent on the problem set, not the points they are associated with.
- In order for your submission to be considered complete, you need to push both your `ps4.qmd` and `ps4.pdf` to your repository. Submissions that do not include both files will automatically receive 50% credit.

```python
import pandas as pd
import altair as alt
import time

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

Step 1: Develop initial scraper and crawler

```python
import requests
from bs4 import BeautifulSoup

url = 'https://oig.hhs.gov/fraud/enforcement/'

response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

ul = soup.find("ul", class_="usa-card-group padding-y-0")
rows = ul.find_all("li")

data = []

for row in rows:
  a = row.find("a")
  if a is None:
        continue

  title = a.text
  link = "https://oig.hhs.gov" + a["href"]
  span = row.find("span", class_="text-base-dark padding-right-105")
  date = span.text
  cat_li = row.find("li", class_="display-inline-block usa-tag
↪  text-no-lowercase text-base-darkest bg-base-lightest margin-right-1")
  category = cat_li.text
  data.append({"title": title, "date": date, "category": category, "link":
↪  link})

df = pd.DataFrame(data)
```

```
display(df.head())
```

	title	date	category	lin
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions	htt
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions	htt
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	COVID-19	htt
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions	htt
4	Former NFL Player Convicted for $197M Medicare...	February 3, 2026	Criminal and Civil Actions	htt

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

Inputs: start_year (int) start_month (int) run_scraper (bool indicator to control whether the scraper actually runs)

If start_year < 2013: print("Reminder: Only enforcement actions after 2013 are listed. Please use year >= 2013.") return empty dataframe

BASE_URL = "https://oig.hhs.gov/fraud/enforcement/" start_date = first day of (start_year, start_month) output_filename = f"enforcement_actions_{start_year}_{start_month}.csv"

IF run_scraper == True: scrape pages, build dataframe, save to CSV, return dataframe ELSE: IF output_filename exists: load dataframe from CSV and return it ELSE: print("Cached file not found. Set run_scraper=True to scrape.")

page = 0 all_results = empty list

WHILE True: If page == 0: page_url = BASE_URL Else: page_url = BASE_URL + "?page=" + str(page) Request and parse HTML for page_url For each actions on the page: * Extract Title (text) * Extract Link (href; convert to absolute URL) * Extract Date (text; convert to datetime) * Extract Category/Type Store as columns: title, date, category, link Filter to keep only rows with date >= start_date Append filtered rows to all_results Stopping condition: Let min_date_on_page be the earliest date found on this page If min_date_on_page < start_date: BREAK page = page + 1 sleep(1)

Concatenate all_results into one dataframe Sort by date (ascending) so the earliest action is at the top

Write dataframe to output_filename Return dataframe

- b. Create Dynamic Scraper

```
BASE = "https://oig.hhs.gov"
BASE_LIST = "https://oig.hhs.gov/fraud/enforcement/"

def scrape_page(page_num):
    """Scrape one enforcement actions list page and return a dataframe."""
    url = BASE_LIST if page_num == 0 else f"{BASE_LIST}?page={page_num}"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'lxml')

    ul = soup.find("ul", class_="usa-card-group padding-y-0")
    if ul is None:
        return pd.DataFrame(columns=["title", "date", "category", "link"])

    rows = ul.find_all("li")

    data = []
    for row in rows:
        a = row.find("a")
        if a is None:
            continue
        title = a.text
        link = BASE + a["href"]
        span = row.find("span", class_="text-base-dark padding-right-105")
        date = span.text
        cat_li = row.find("li", class_="display-inline-block usa-tag
↪   text-no-lowercase text-base-darkest bg-base-lightest margin-right-1")
        category = cat_li.text

        data.append({"title": title, "date": date, "category": category,
↪   "link": link})

    df = pd.DataFrame(data)
    df["date"] = pd.to_datetime(df["date"], errors="coerce")
    return df
```

```
def enforcement_actions_since(start_year, start_month, run_scraper: bool =
↪   False):
    """
    Crawl actions from start_year/start_month up to today.
    If run_scraper=False: load cached CSV if exists, otherwise return
↪   message.
    If run_scraper=True: scrape and save to CSV.
    """
    if start_year < 2013:
        print("Only enforcement actions after 2013 are listed. Please use
         ↪  year >= 2013.")
        return pd.DataFrame(columns=["title", "date", "category", "link"])

    out_file = f"enforcement_actions_{start_year}_{start_month}.csv"

    if not run_scraper:
        try:
            df_cached = pd.read_csv(out_file)
            df_cached["date"] = pd.to_datetime(df_cached["date"],
↪   errors="coerce")
            return df_cached
        except FileNotFoundError:
            print(f"Cached file not found: {out_file}. Set run_scraper=True
             ↪  to scrape.")
            return pd.DataFrame(columns=["title", "date", "category",
             ↪  "link"])

    start_date = pd.Timestamp(start_year, start_month, 1)

    all_kept = []
    page = 0

    while True:
        df_page = scrape_page(page)
        if df_page.empty:
            break

        kept = df_page[df_page["date"] >= start_date].copy()
        all_kept.append(kept)

        min_date_on_page = df_page["date"].min()
        if pd.notna(min_date_on_page) and min_date_on_page < start_date:
            break

        page += 1
        time.sleep(1)

    if all_kept:
      df_all = pd.concat(all_kept, ignore_index=True)
    else:
      df_all =
↪   pd.DataFrame(columns=["title","date","category","link"])(columns=["title","date","catego
    df_all = df_all.sort_values("date").reset_index(drop=True)

    df_all.to_csv(out_file, index=False)
    return df_all
```

```
df_202401 = enforcement_actions_since(2024, 1, run_scraper=False)#Turn
↪   run_scraper to False for knitting.
```

```
print(len(df_202401))
earliest = df_202401.sort_values("date").iloc[0]
display(earliest)
```

```
1807

title       Former Nurse Aide Indicted In Death Of Clarksv...
date                                     2024-01-03 00:00:00
category                           State Enforcement Agencies
link        https://oig.hhs.gov/fraud/enforcement/former-n...
Name: 0, dtype: object
```

I got 1807 enforcement actions in the the final dataframe. The earliest enforcement action took place on 01/03/2024 and it is called Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia. It belongs to State Enforcement Agencies.

- c. Test Your Code

```
df_202201 = enforcement_actions_since(2022, 1, run_scraper=False)#Turn
↪   run_scraper to False for knitting.
```

```
print(len(df_202201))
earliest_2022 = df_202201.sort_values("date").iloc[0]
display(earliest_2022)
```

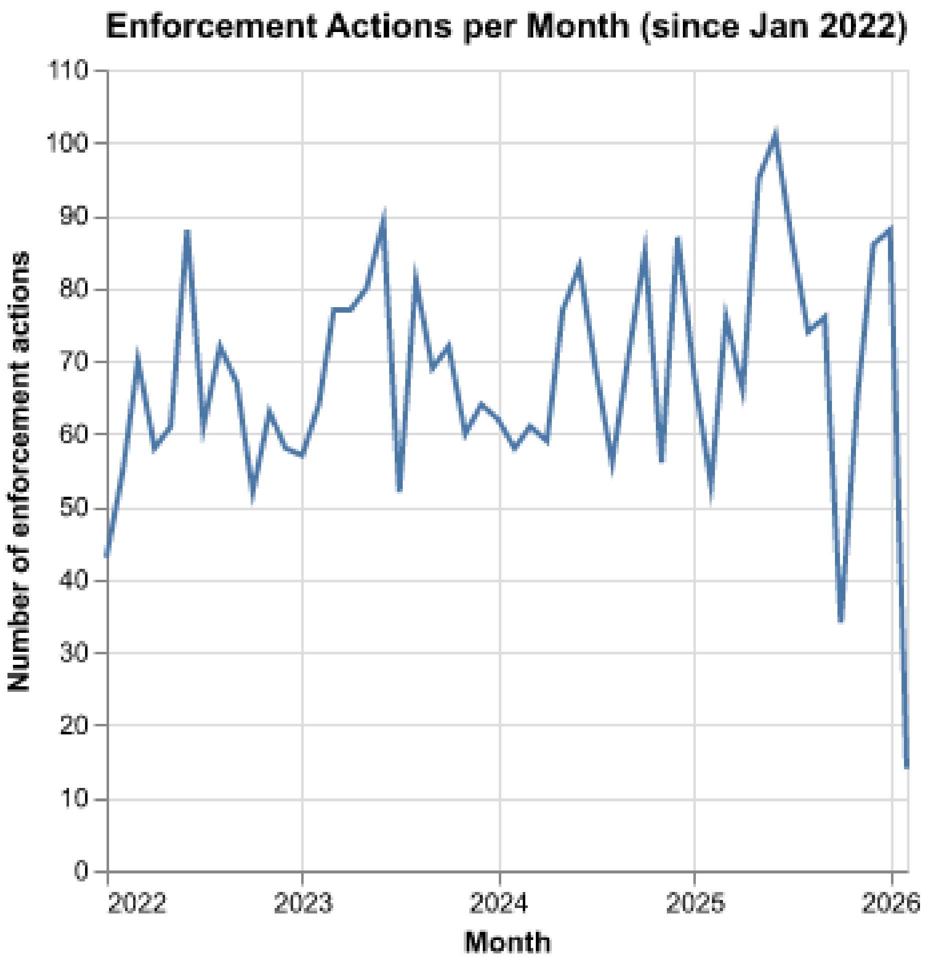
```
3397

title       Integrated Pain Management Medical Group Agree...
date                                     2022-01-04 00:00:00
category                              Fraud Self-Disclosures
link        https://oig.hhs.gov/fraud/enforcement/integrat...
Name: 0, dtype: object
```

I got 3397 enforcement actions in the the final dataframe. The earliest enforcement action took place on 01/04/2022 and it is called Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals. It belongs to Fraud Self-Disclosures.

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

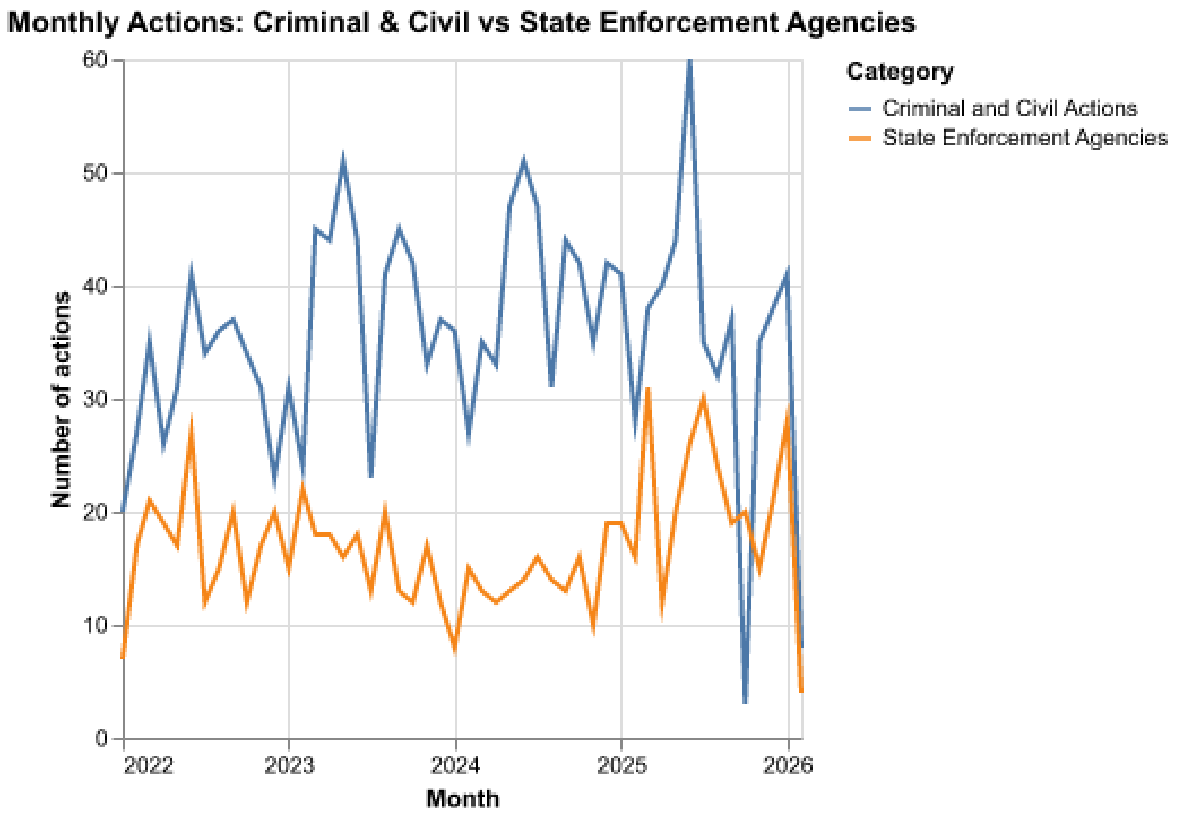
```
df_202201["month"] = df_202201["date"].dt.to_period("M").dt.to_timestamp()

monthly_total = (
    df_202201.groupby("month")
      .size()
      .reset_index(name="counts")
)

chart1 = (
    alt.Chart(monthly_total, title="Enforcement Actions per Month (since Jan
 ↪  2022)")
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("counts:Q", title="Number of enforcement actions"),
    )
)

chart1
```



2. Plot the number of enforcement actions categorized:

- based on "Criminal and Civil Actions" vs. "State Enforcement Agencies"

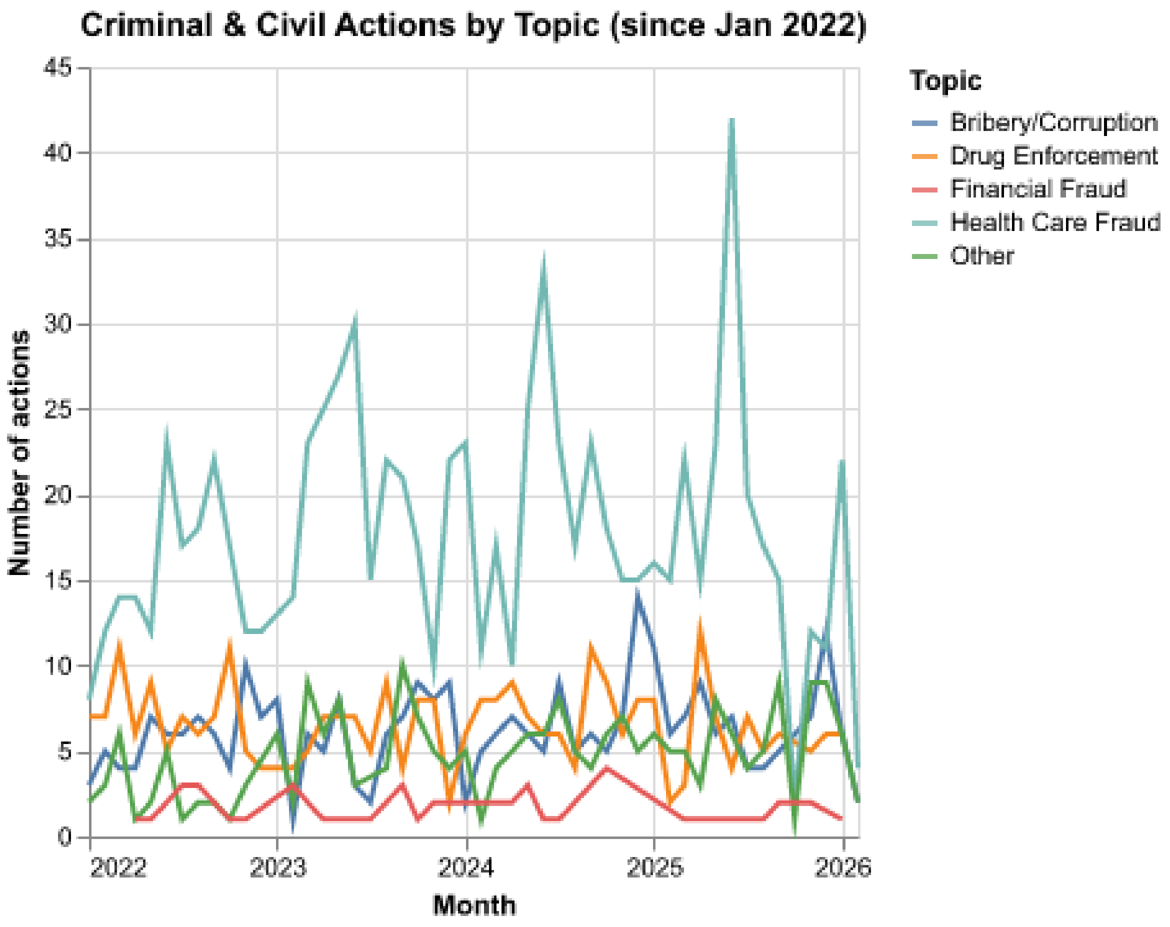
```
keep = ["Criminal and Civil Actions", "State Enforcement Agencies"]
df_2 = df_202201[df_202201["category"].isin(keep)].copy()

monthly_2 = (
    df_2.groupby(["month", "category"])
          .size()
          .reset_index(name="counts")
)

chart2a = (
    alt.Chart(monthly_2, title="Monthly Actions: Criminal & Civil vs State
↪   Enforcement Agencies")
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("counts:Q", title="Number of actions"),
        color=alt.Color("category:N", title="Category"),
    )
)

chart2a
```



- based on five topics

```
cc = df_202201[df_202201["category"] == "Criminal and Civil Actions"].copy()
t = cc["title"].str.lower()

def classify_topic(title_lower):
    drug_kw = ["drug", "opioid", "fentanyl", "pill", "pharmacy",
↪   "prescription", "controlled substance", "suboxone"]
    if any(k in title_lower for k in drug_kw):
        return "Drug Enforcement"

    bribery_kw = ["brib", "kickback", "corrupt", "conspiracy", "bribe",
↪   "money laundering"]
    if any(k in title_lower for k in bribery_kw):
        return "Bribery/Corruption"

    financial_kw = ["bank", "financial", "wire", "launder", "tax",
↪   "securities", "investment", "credit", "loan"]
    if any(k in title_lower for k in financial_kw):
        return "Financial Fraud"

    health_kw = ["medicare", "medicaid", "health care", "healthcare",
↪   "hospital", "clinic", "physician", "doctor", "billing", "claims", "dme"]
    if any(k in title_lower for k in health_kw):
        return "Health Care Fraud"

    return "Other"

cc["topic"] = t.apply(classify_topic)
```

```
monthly_topic = (
    cc.groupby(["month", "topic"])
      .size()
      .reset_index(name="counts")
)

chart2b = (
    alt.Chart(monthly_topic, title="Criminal & Civil Actions by Topic (since
↪   Jan 2022)")
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("counts:Q", title="Number of actions"),
        color=alt.Color("topic:N", title="Topic"),
    )
)

chart2b
```



Criminal & Civil Actions by Topic (since Jan 2022)
Number of actions
0
5
10
15
20
25
30
35
40
45
2022
2023
2024
2025
2026
Month
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other